PATENT ABSTRACTS OF JAPAN

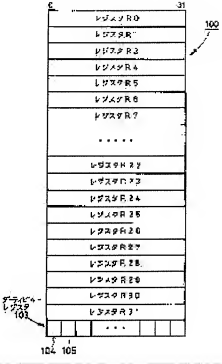
(11)Publication number : 09-212371
(43)Date of publication of application : 15.08.1997

(51)Int.Cl. G06F 9/46

(21)Application number : 08-020338 (71)Applicant : NEC CORP
(22)Date of filing : 07.02.1996 (72)Inventor : CHIBA MASAKAZU

(54) REGISTER SAVING AND RESTORING SYSTEM

(57)Abstract:
PROBLEM TO BE SOLVED: To reduce the overhead of an OS by shortening the time required for the saving processing of the contents of registers to a context area at the time of task switching to the minimum in a microprocessor for executing multitask processing.
SOLUTION: Dirty bits 104, 105 are prepared correspondingly to plural registers 100, and when the contents of the register 100 are changed, its corresponding bit is turned to '1'. When there is no change, the corresponding bit is turned to '0'. In the case of saving the contents of respective registers 100 to a context area when task switching is generated, respective bits are referred to, and when the bits are '0', a saving instruction is not executed. Namely, saving for the contents of the registers 100 and the updating of storing addresses for saving are not executed. Thereby, the processing time can be shortened.



LEGAL STATUS

[Date of request for examination] 07.02.1996
[Date of sending the examiner's decision of rejection] 08.09.1998
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]
[Date of final disposal for application]
[Patent number]
[Date of registration]
[Number of appeal against examiner's decision of rejection]
[Date of requesting appeal against examiner's decision of rejection]
[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報(A)　(11)特許出願公開番号

特開平9－212371

(43)公開日　平成9年(1997)8月15日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　9/46	３１３		Ｇ０６Ｆ　9/46	３１３Ａ	

審査請求　有　請求項の数6　ＯＬ　（全７頁）

(21)出願番号　特願平8－20338

(22)出願日　平成8年(1996)2月7日

(71)出願人　000004237
日本電気株式会社
東京都港区芝五丁目７番１号

(72)発明者　千葉　雅一
東京都港区芝五丁目７番１号　日本電気株式会社内

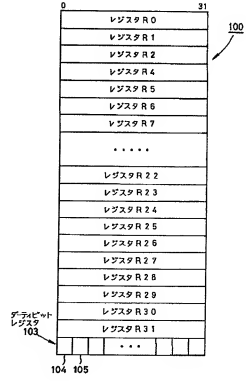
(74)代理人　弁理士　▲柳▼川　信

(54)【発明の名称】　レジスタ退避及び復元システム

(57)【要約】　（修正有）

【課題】　マルチタスク処理を行うマイクロプロセッサにおいて、タスク切替え時におけるレジスタの内容のコンテキスト領域への退避処理に要する時間を極力少なくして、ＯＳのオーバーヘッドを減少させる。

【解決手段】　複数のレジスタ１００に夫々対応して、ダーティビット１０４，１０５を設け、レジスタの内容が変化したときに対応ビットを“１”とし、変化しないときは“０”とする。タスク切替えが発生したときに各レジスタ内容をコンテキスト領域へ退避する場合、各ビットを参照して、“０”であれば、退避命令を実行しない。すなわち、レジスタの内容の退避も退避のための格納アドレスの更新も行わない。よって、それだけ処理時間が少なくなる。



1

【特許請求の範囲】

【請求項１】　複数のレジスタを有し、これ等レジスタの格納内容をメモリ上の退避領域に退避するレジスタ退避システムであって、前記レジスタの各々に対応して設けられ対応レジスタの内容が変化したことを示すビット群と、前記レジスタの内容の退避時に、前記ビット群を参照してそのビットが内容の変化を示すときは前記退避領域への格納アドレスの更新を行って対応レジスタの内容の退避を行い、内容の変化を示さないときは前記格納アドレスの更新及び対応レジスタの退避を行わないよう制御する手段とを含むことを特徴とするレジスタ退避システム。

【請求項２】　前記退避時に、前記ビット群の内容を全て退避するようにしたことを特徴とする請求項１記載のレジスタ退避システム。

【請求項３】　複数のレジスタを有し、これ等レジスタの格納内容をメモリ上の退避領域に退避し、また退避領域の退避中の内容を前記レジスタに復元するようにしたレジスタ退避及び復元システムであって、前記レジスタの各々に対応して設けられ対応レジスタの内容が変化したことを示すビット群と、前記レジスタの内容の退避時に、前記ビット群を参照してそのビットが内容の変化を示すときは前記退避領域への格納アドレスの更新を行って対応レジスタの内容の退避を行い、内容の変化を示さないときは前記格納アドレスの更新及び対応レジスタの退避を行わないよう制御する手段と、前記退避時に前記ビット群の内容を全て退避するよう制御する手段と、前記退避領域の退避内容の前記レジスタへの復元時に、前記退避中のビット群を参照してそのビットが内容の変化を示すときは前記退避領域から読出すべき復元アドレスの更新を行って対応復元データの対応レジスタへの復元を行い、内容の変化を示さないときは前記復元アドレスの更新及び対応レジスタへの復元を行わないよう制御する手段とを含むことを特徴とするレジスタ退避及び復元システム。

【請求項４】　前記復元の終了に応答して前記レジスタの各々に対応して設けられたビット群を全てクリアするようにしたことを特徴とする請求項３記載のレジスタ退避及び復元システム。

【請求項５】　前記退避中のビット群の参照に際して、前記退避中のビット群の内容を順次ビットシフトしつつシフトアウトしたビットの内容を参照するようにしたことを特徴とする請求項３記載のレジスタ退避及び復元システム。

【請求項６】　前記ビット群の内容が全てクリア状態になった時に前記復元処理を終了するようにしたことを特徴とする請求項５記載のレジスタ退避及び復元システム。

【発明の詳細な説明】

【０００１】

2

【発明の属する技術分野】本発明はレジスタ退避及び復元システムに関し、特にマルチタスク処理を行うＯＳ（オペレーティングシステム）を使用したマイクロプロセッサにおけるタスク切替え時レジスタ内容の退避／復元処理の制御を行うレジスタ退避及び復元システムに関するものである。

【０００２】

【従来の技術】現在、マイクロプロセッサを効率良く使用するために、マルチプログラミングの方式が用いられている。マルチプログラミング方式では、プログラムを実行単位であるタスクに分割し、ＯＳが種々の状況に応じてタスクを選択し、ＣＰＵの実行権を割り当てている。ＯＳがあるタスクにＣＰＵの実行権を割り当てる際には、現在ＣＰＵが使用しているタスクのコンテキストをメモリ上の格納領域に退避し、割り当てるタスクのコンテキスト領域からコンテキストを復元する。この時の動作を図６，７を参照しながら説明する。

【０００３】図６はＣＰＵのレジスタの例を示したものである。このＣＰＵは３２ビット長のレジスタを３２本持つものである。また、ＣＰＵはこれ等のレジスタの退避及び復元を命令で実行するものとする。

【０００４】図７はＣＰＵの実行権を移す際のフローを示したものである。今、タスクは簡単のためにタスクＡとタスクＢの２つとする。初めはタスクＡにＣＰＵの実行権があり、タスクＡからタスクＢにＣＰＵの実行権が移るものとする。また、タスクのコンテキストを図６に示したレジスタとし、コンテキストの格納領域はタスクコントロールブロック内に確保されているものとする。

【０００５】ＣＰＵの実行権を切り換えるために（ステップ７０１）、ＯＳはタスクＡのタスクコントロールブロックアドレスを得る（ステップ７０２）。この時ＯＳはこのタスクコントロールブロック内にタスクＡのコンテキストを格納するために、レジスタの退避命令を実行する。この命令によってＣＰＵは３２本のレジスタの内容をレジスタＲ０から順にレジスタＲ３１まで全てを退避せしめてメモリ上の退避領域に格納する（ステップ７０３）。

【０００６】次に、メモリ上の他の領域における実行待ちの行列の先頭から、タスクＢのタスクコントロールブロックアドレスを得る（ステップ７０４）。ＯＳはこのタスクコントロールブロック内のタスクＢのコンテキストをＣＰＵのレジスタに格納するために、レジスタの復帰命令を実行する。この命令によってＣＰＵはタスクコントロールブロック内に格納してあるデータをレジスタＲ０からＲ３１に格納する（ステップ７０５）。この様にして、ＯＳはタスクＡからタスクＢにＣＰＵのコンテキストを入れ替える。

【０００７】別の従来例を説明する。この例は特開平４－２１１８３７号公報に開示のものであり、この従来例では、前述の従来例同様に、ＯＳはレジスタの内容を退

避させるための退避命令を実行する。ＣＰＵはレジスタＲ０から順に退避格納するが、この際各レジスタに対応して予め設けられているモディファイビットの状態を参照する。

【０００８】モディファイビットとは実行中のタスク内でレジスタの内容に変更があった場合に［１］がセットされるビットのことである。

【０００９】先ず、ＣＰＵはレジスタＲ０に対応したモディファイビットをチェックする。この場合、モディファイビットが［１］であったとする。よって、このレジスタＲ０の内容は変更されているので、ＣＰＵはレジスタＲ０の内容をコンテキスト格納領域に退避格納し、次のレジスタＲ１のための格納アドレスを更新する。

【００１０】次に、レジスタＲ１のモディファイビットをチェックする。この場合、レジスタＲ１のモディファイビットが［０］であったとする。よって、このレジスタＲ１の内容は変更されていないので、レジスタＲ１の内容は格納せず、格納アドレスのみを更新する。

【００１１】この様に、モディファイビットが［１］の場合は、ＣＰＵは対応するレジスタの内容をコンテキスト格納領域に格納し、格納アドレスを更新し、モディファイビットが［０］の場合は、ＣＰＵは対応するレジスタの内容を格納せず、格納アドレスのみを更新する。以上のように、ＣＰＵはレジスタの退避を行う。

【００１２】次に、ＯＳはタスクＢのタスクコントロールブロックアドレスを得る。ＯＳはタスクコントロールブロックに格納してあるタスクＢのコンテキストをＣＰＵのレジスタに格納するために、レジスタの復元命令を実行する。

【００１３】この命令によってＣＰＵは、モディファイビットを全て［０］にし、コンテキスト格納領域に格納してあるデータを全てのレジスタに格納する。この様にして、ＯＳはタスクＡからタスクＢにコンテキストを入れ替える。

【００１４】

【発明が解決しようとする課題】マイクロプロセッサは内部におけるコンテキストの転送を非常に高速に行う。しかし、マルチタスク処理を行うマイクロプロセッサにおいては、タスクを入れ替える度にレジスタの内容をメモリに退避格納し、次に実行するタスクのコンテキストをレジスタに復元する必要がある。

【００１５】その際に、マイクロプロセッサの外部バスを使用するためにマイクロプロセッサ内部のように高速にデータ転送が行えない。マイクロプロセッサを使用したシステムにおいて、外部アクセスを用いるアクセスが生じると性能が低下する。

【００１６】また、先の特開平４－２１１８３７号公報のモディファイビットを用いたレジスタシステムにおいても、レジスタの本数分の退避領域への格納アドレスを更新し、復元の際にはレジスタの本数分コンテキストをレジスタに戻すために、外部アクセス時間がＯＳの処理時間内のオーバーヘッドとなってしまう。

【００１７】本発明の目的は、マルチタスクの切替え時におけるレジスタ内容の退避／復元処理のオーバーヘッドを極力少なくするようにしたレジスタ退避復元システムを提供することである。

【００１８】

【課題を解決するための手段】本発明によるレジスタ退避システムは、複数のレジスタを有し、これ等レジスタの格納内容をメモリ上の退避領域に退避するレジスタ退避システムであって、前記レジスタの各々に対応して設けられ対応レジスタの内容が変化したことを示すビット群と、前記レジスタの内容の退避時に、前記ビット群を参照してそのビットが内容の変化を示すときは前記退避領域への格納アドレスの更新を行って対応レジスタの内容の退避を行い、内容の変化を示さないときは前記格納アドレスの更新及び対応レジスタの退避を行わないよう制御する手段とを含むことを特徴としている。

【００１９】そして、前記退避時に、前記ビット群の内容を全て退避するようにしたことを特徴としている。

【００２０】また、本発明によるレジスタ退避及び復元システムは、複数のレジスタを有し、これ等レジスタの格納内容をメモリ上の退避領域に退避し、また退避領域の退避中の内容を前記レジスタに復元するようにしたレジスタ退避及び復元システムであって、前記レジスタの各々に対応して設けられ対応レジスタの内容が変化したことを示すビット群と、前記レジスタの内容の退避時に、前記ビット群を参照してそのビットが内容の変化を示すときは前記退避領域への格納アドレスの更新を行って対応レジスタの内容の退避を行い、内容の変化を示さないときは前記格納アドレスの更新及び対応レジスタの退避を行わないよう制御する手段と、前記退避領域の退避内容の前記レジスタへの復元時に、前記退避中のビット群を参照してそのビットが内容の変化を示すときは前記退避領域から読出すべき復元アドレスの更新を行って対応復元データの対応レジスタへの復元を行い、内容の変化を示さないときは前記復元アドレスの更新及び対応レジスタへの復元を行わないよう制御する手段とを含むことを特徴としている。

【００２１】そして、前記復元の終了に応答して、前記レジスタの各々に対応して設けられたビット群を全てクリアするようにしたことを特徴としている。

【００２２】このビット群クリア処理をなす代わりに、前記退避中のビット群の参照に際して、前記退避中のビット群の内容を順次ビットシフトしつつシフトアウトしたビットの内容を参照し、前記ビット群の内容が全てクリア状態になった時に前記復元処理を終了するようにしたことを特徴としている。

【００２３】

5

【発明の実施の形態】本発明の作用について述べる。レジスタの内容に変化がない場合には、退避や復元の必要もないために、変化のあったレジスタの内容のみを退避や復元処理すると共に、その時のみメモリ上の退避領域への退避格納アドレスの更新や復元アドレスの更新を行うことで、OSのオーバーヘッドを極力少なくすることが可能となる。

【0024】本発明の実施例を図面を参照しながら説明する。

【0025】図1は本発明の一実施例に用いるレジスタシステム構成を示したものである。図で使用しているレジスタシステム100は、一つのレジスタが32ビット構成であり、32本で構成されている。

【0026】また、一般使用を目的とした32本のレジスタとは別に、各レジスタの変化を示す32個のダーティビットを並べたダーティビット群レジスタ103を有する。このダーティビットは、例えば、レジスタR0やR1に対して夫々ダーティビット104や105というように、各レジスタに対して1対1に用意される。

【0027】そして、タスクAを実行中にレジスタR0の内容を変更した場合、レジスタR0に対応するダーティビット104が[1]にセットされる。また、タスクAの最中にレジスタR0の内容が変化しなかった場合、ダーティビット104の[0]の値は変化しない。同様に、全てのダーティビットの値がタスクAの実行中に決定される。

【0028】タスクAからタスクBに変更する際、コンテキストを入れ替える必要があるが、OSは図2に示す退避格納アルゴリズムに従ってレジスタの内容を退避する退避命令を実行することでこれを行う（ステップ301）。CPUはレジスタR0の内容から順に退避格納するが、この際に各レジスタに対応したダーティビットを参照する。先ず、CPUはレジスタR0に対応したダーティビットをチェックする（ステップ302，303）。

【0029】この例では、ダーティビット104の値を[1]、ダーティビット105の値を[0]とする。よってCPUはレジスタR0の内容をコンテキスト格納領域に格納し（ステップ305）、格納アドレスを更新する（ステップ306）。

【0030】次に、CPUはレジスタR1のダーティビットをチェックする。レジスタR1のダーティビットは[0]である。よってレジスタR1の内容は格納せず、格納アドレスも更新しない。

【0031】以上のようにレジスタR31までの処理を繰り返し（ステップ307，308）、CPUはレジスタの退避を行う。レジスタR31の処理の後、ダーティビット群のレジスタ103の内容もコンテキスト格納領域に格納し（ステップ309）、格納アドレスを更新する（ステップ310）。

6

【0032】次にOSは、タスクBのタスクコントロールブロックアドレスを得る。この時、タスクBのコンテキスト格納領域の内容は図3に示す如くであるとする。OSはタスクコントロールブロックに格納してあるタスクBのコンテキスト200をCPUのレジスタに格納するために、レジスタの復元命令を実行する。この命令によってCPUは、先ずコンテキスト格納領域に格納してあるタスクBのダーティビット群のレジスタ（図1の103）に格納する。

【0033】つまりこのレジスタ203は、タスクBのコンテキストの内、コンテキスト格納領域から復元すべきコンテキスト内容を示している。

【0034】タスクBからの他のタスクに切り替わるときに、コンテキスト格納領域に退避格納した値が図3に示す如くなっているわけであり、つまりタスクBの最中に変化のなかったレジスタに関しては対応するダーティビットが[0]となり、格納の際に格納アドレスを更新していないために、ダーティビットが[1]の値のレジスタの内容のみが格納されていることになる。

【0035】図3では、レジスタのR0，R2の各内容が格納されていて、それに対応するダーティビット204，206は夫々[1]になっている。また、レジスタR1の内容は格納されておらず、よってそれに対応するダーティビット205は[0]になっている。

【0036】この様に本発明の実施例では、タスク実行中に変更のあったコンテキストのみをコンテキスト格納領域に退避格納している。

【0037】次に復元命令によって、タスクのコンテキストが入れ替わる様子を図4の復元アルゴリズムに従って説明する。CPUが復元命令を実行すると（ステップ401）、先ず、コンテキスト格納領域に格納された32ビットのダーティビット群203をCPU内のダーティビット用の32ビットのダーティビットレジスタ103に格納する（ステップ402）。

【0038】次に、CPUはコンテキスト格納領域から32ビットずつレジスタの内容を読出し格納するが、その際は先に格納したダーティビットレジスタ103の内容を参照し（ステップ403，404）、ダーティビットが[1]になっているレジスタのみに格納を行い（ステップ405，406）、復元アドレスを更新する（ステップ407）。全てのダーティビットのチェックが終了すると（ステップ408，409）、ダーティビット群103を全て[0]クリアする（ステップ410）。

【0039】この様にしてOSは、タスクのコンテキストを入れ替える（ステップ411）。

【0040】次に、図5を用いて本発明の他の実施例を説明する。尚、図5において、図4と同等ステップは同一符号を付してその説明を省略する。

【0041】先の実施例である図4に示した復元アルゴリズムでは、タスクのコンテキストをコンテキスト格納

領域から復元する際は、ダーティビットレジスタの内容を基に各レジスタにコンテキストを格納し、その後全てのダーティビットをチェックする際に、ダーティビットレジスタ（図1の103）の内容を1ビットずつ左にシフトし（ステップ503）、キャリーアウトした値をチェックするようにする（ステップ505）。この時、最下位ビットには1ビットの左シフトの度に0を右から詰めていく（ステップ504）。

【0042】また、左シフト動作の直後に毎回ダーティビットレジスタの値が全て［0］になっていないかをチェックする（ステップ508）。コンテキストの復元の最中にダーティビットレジスタの内容が全て［0］になったら、コンテキストの復元命令は終了する。

【0043】本実施例では、コンテキストを復元する命令の最後のダーティビットの0クリアが必要ないことと、ダーティビットのチェックの順番で最後の［1］をチェックした後の余分なチェックを省略することができる。

【0044】尚、本発明のより良い理解のために、追加的に説明を加える。コンテキストの切替えが発生すると、レジスタ（汎用レジスタ）の内容をメモリ上に退避するが、この時、メモリ上の退避箇所のアドレスが必要となる。従来においては、ストア命令等を用いてメモリ上の退避領域へ格納して行くが、その際のアドレスはソフトウェア上で更新しながら行われる。

【0045】つまり、上述した32本のレジスタが存在するようなハードウェアの場合、32回のアドレス更新を行うストア命令を実行して退避が行われることになる。

【0046】しかし、本発明では、メモリ上の格納場所を節約するためにも、内容が変化していないレジスタに関するストア命令を実行しないようにしたものであり、よって内容が変化しないレジスタに関しては、退避及び＊そのアドレス更新も行わないようにしているのである。

【0047】

【発明の効果】以上述べた如く、本発明によれば、内容に変更のあったレジスタのみをコンテキスト格納領域に格納し、復元する際も変更のあったレジスタの内容のみを復元しているために、余計なメモリアクセス、つまり外部バスのアクセスが必要最小限になり、従って、OSの処理時間内のオーバーヘッドを最小に押さえられるため、コンテキストの変更によるOS性能のロスを最小限に押さえることができるという効果がある。

【図面の簡単な説明】

【図1】本発明の実施例におけるレジスタの構成を示す図である。

【図2】本発明の実施例の退避処理動作を示すフローチャートである。

【図3】本発明の実施例におけるタスクBのコンテキスト格納領域内の内容を示す図である。

【図4】本発明の実施例の復元処理動作の一例を示すフローチャートである。

【図5】本発明の実施例の復元処理動作の他の例を示すフローチャートである。

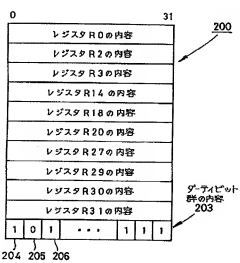
【図6】従来例を説明するためのレジスタの構成図である。

【図7】従来のコンテキスト入れ替え処理時の動作を示すフローチャートである。

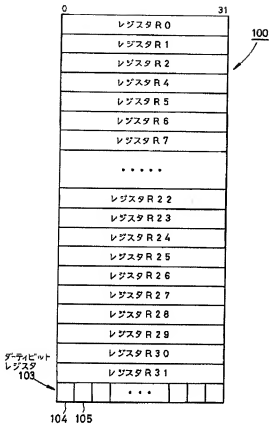
【符号の説明】

100　レジスタ
103　ダーティビットレジスタ
104，105　ダーティビット
200　コンテキスト格納領域の内容
203　ダーティビット群の内容
204～206　ダーティビット

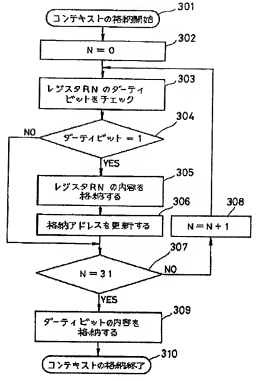
【図3】



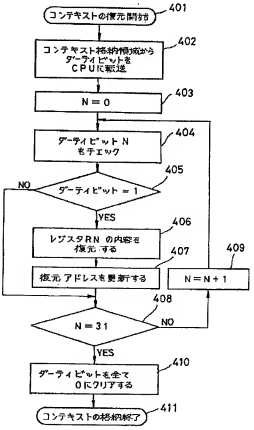
【図１】


0
31
レジスタＲ０
レジスタＲ１
レジスタＲ２
レジスタＲ４
レジスタＲ５
レジスタＲ６
レジスタＲ７
・・・・・
レジスタＲ２２
レジスタＲ２３
レジスタＲ２４
レジスタＲ２５
レジスタＲ２６
レジスタＲ２７
レジスタＲ２８
レジスタＲ２９
レジスタＲ３０
レジスタＲ３１
・・・
100
ダーティビット
レジスタ
103
104
105

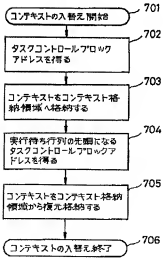
【図２】


コンテキストの格納開始
301
N＝0
302
レジスタRN のダーティ
ビットをチェック
303
ダーティビット＝１
304
NO
YES
レジスタRN の内容を
格納する
305
格納アドレスを更新する
306
N＝N＋１
308
N＝３１
307
NO
YES
ダーティビットの内容を
格納する
309
コンテキストの格納終了
310

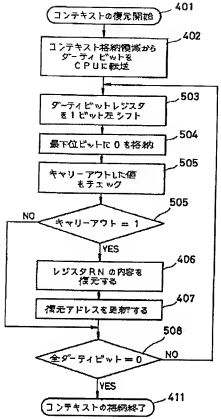
【図４】


コンテキストの復元開始
401
コンテキスト格納領域から
ダーティビットを
CPUに転送
402
N＝0
403
ダーティビット N
をチェック
404
ダーティビット＝１
405
NO
YES
レジスタRN の内容を
復元する
406
復元アドレスを更新する
407
N＝N＋１
409
N＝３１
408
NO
YES
ダーティビットを全て
０にクリアする
410
コンテキストの格納終了
411

【図７】


コンテキストの入替え開始
701
タスクコントロールブロック
アドレスを得る
702
コンテキストをコンテキスト格
納領域へ格納する
703
実行待ち行列の先頭になる
タスクコントロールブロックア
ドレスを得る
704
コンテキストをコンテキスト格納
領域から復元格納する
705
コンテキストの入替え終了
706

【図５】



【図６】

0	31
レジスタＲ０	
レジスタＲ１	
レジスタＲ２	
レジスタＲ４	
レジスタＲ５	
レジスタＲ６	
レジスタＲ７	
・・・・・	
レジスタＲ２２	
レジスタＲ２３	
レジスタＲ２４	
レジスタＲ２５	
レジスタＲ２６	
レジスタＲ２７	
レジスタＲ２８	
レジスタＲ２９	
レジスタＲ３０	
レジスタＲ３１	